

Locality failover

 4 minute read  page test

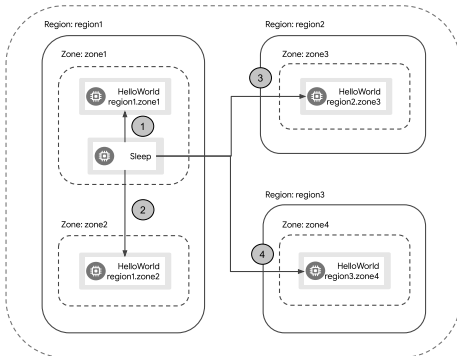
Follow this guide to configure your mesh for locality failover.

Before proceeding, be sure to complete the steps under [before you begin](#).

In this task, you will use the `Sleep pod` in `region1.zone1` as the source of requests to the `HelloWorld` service.

You will then trigger failures that will cause failover between localities in the following sequence:

Mesh: mesh1



Locality failover sequence

Internally, Envoy priorities are used to control failover. These priorities will be assigned as follows for traffic originating from the `sleep` pod (in `region1 zone1`):

Priority	Locality	Details
0	<code>region1. zone1</code>	Region, zone, and sub-zone all match.
1	None	Since this task doesn't use

		sub-zones, there are no matches for a different sub-zone.
2	region1. zone2	Different zone within the same region.
3	region2. zone3	No match, however failover is defined for region1->region2.
4	region3. zone4	No match and no failover defined for region1->region3.

Configure locality failover

Apply a `DestinationRule` that configures the following:

- Outlier detection for the `HelloWorld` service. This is required in order for failover to function properly. In particular, it configures the sidecar proxies to know when endpoints for a service are unhealthy, eventually triggering a failover to the next locality.
- Failover policy between regions. This ensures that failover beyond a region boundary will behave predictably.

- Connection Pool policy that forces each HTTP request to use a new connection. This task utilizes Envoy's drain function to force a failover to the next locality. Once drained, Envoy will reject new connection requests. Since each request uses a new connection, this results in failover immediately following a drain. **This configuration is used for demonstration purposes only.**

```
$ kubectl --context="${CTX_PRIMARY}" apply -n sample -f - <<EOF
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: helloworld
```

```
spec:
  host: helloworld.sample.svc.cluster.local
  trafficPolicy:
    connectionPool:
      http:
        maxRequestsPerConnection: 1
    loadBalancer:
      simple: ROUND_ROBIN
      localityLbSetting:
        enabled: true
        failover:
          - from: region1
            to: region2
    outlierDetection:
      consecutive5xxErrors: 1
      interval: 1s
      baseEjectionTime: 1m
```

EOF

Verify traffic stays in region1.zone1

Call the `HelloWorld` service from the `Sleep` pod:

```
$ kubectl exec --context="${CTX_R1_Z1}" -n sample -c sleep \
  "$ (kubectl get pod --context="${CTX_R1_Z1}" -n sample -l \
    app=sleep -o jsonpath='{.items[0].metadata.name}')" \
  -- curl -sSL helloworld.sample:5000/hello
Hello version: region1.zone1, instance: helloworld-region1.zone1
-86f77cd7b-cpxhv
```

Verify that the `version` in the response is `region1.zone`.

Repeat this several times and verify that the response is always the same.

Failover to `region1.zone2`

Next, trigger a failover to `region1.zone2`. To do this, **you** drain the Envoy sidecar proxy **for** `HelloWorld` in `region1.zone1`:

```
$ kubectl --context="${CTX_R1_Z1}" exec \
    "$(kubectl get pod --context="${CTX_R1_Z1}" -n sample -l app=hello-world \
    -l version=region1.zone1 -o jsonpath='{.items[0].metadata.name}' )" \
    -n sample -c istio-proxy -- curl -sSL -X POST 127.0.0.1:15000/drain_listeners
```

Call the HelloWorld service from the Sleep pod:

```
$ kubectl exec --context="${CTX_R1_Z1}" -n sample -c sleep \
    "$(kubectl get pod --context="${CTX_R1_Z1}" -n sample -l \
    app=sleep -o jsonpath='{.items[0].metadata.name}')" \
    -- curl -sSL helloworld.sample:5000/hello
Hello version: region1.zone2, instance: helloworld-region1.zone2-86f77cd7b-cpxhv
```

The first call will fail, which triggers the failover. Repeat the command several more times and verify that the `version` in the response is always `region1.zone2`.

Failover to `region2.zone3`

Now trigger a failover to `region2.zone3`. As you did previously, configure the `HelloWorld` in `region1.zone2` to fail when called:

```
$ kubectl --context="${CTX_R1_Z2}" exec \
  "$(kubectl get pod --context="${CTX_R1_Z2}" -n sample -l app=hello-world \
    -l version=region1.zone2 -o jsonpath='{.items[0].metadata.name}'" \
  -n sample -c istio-proxy -- curl -sSL -X POST 127.0.0.1:15000/drain_listeners
```

Call the HelloWorld service from the Sleep pod:

```
$ kubectl exec --context="${CTX_R1_Z1}" -n sample -c sleep \
  "$(kubectl get pod --context="${CTX_R1_Z1}" -n sample -l \
    app=sleep -o jsonpath='{.items[0].metadata.name}')" \
  -- curl -sSL helloworld.sample:5000/hello
Hello version: region2.zone3, instance: helloworld-region2.zone3-86f77cd7b-cpxhv
```

The first call will fail, which triggers the failover. Repeat the command several more times and verify that the `version` in the response is always `region2.zone3`.

Failover to `region3.zone4`

Now trigger a failover to `region3.zone4`. As you did previously, configure the `HelloWorld` in `region2.zone3` to fail when called:

```
$ kubectl --context="${CTX_R2_Z3}" exec \
  "$(kubectl get pod --context="${CTX_R2_Z3}" -n sample -l app=hello-world \
    -l version=region2.zone3 -o jsonpath='{.items[0].metadata.name}'" \
  -n sample -c istio-proxy -- curl -sSL -X POST 127.0.0.1:15000/drain_listeners
```

Call the HelloWorld service from the Sleep pod:

```
$ kubectl exec --context="${CTX_R1_Z1}" -n sample -c sleep \
  "$(kubectl get pod --context="${CTX_R1_Z1}" -n sample -l \
    app=sleep -o jsonpath='{.items[0].metadata.name}')" \
  -- curl -sSL helloworld.sample:5000/hello
Hello version: region3.zone4, instance: helloworld-region3.zone4-86f77cd7b-cpxhv
```

The first call will fail, which triggers the failover.
Repeat the command several more times and verify that the `version` in the response is always `region3.zone4`.

Congratulations! You successfully configured locality failover!

Next steps

Cleanup resources and files from this task.

